华东师范大学计算机科学与技术学院实验报告

课程名称：数据挖掘	**年级**：2020	**实践作业成绩**：
指导教师：兰曼	**姓名**：张宏伟	**提交作业日期**：2023/5/11
实践编号：1	**学号**：10205102417	**实践作业编号**：1

一、 实验名称：新闻标题分类

Web 数据的采集，对爬取到的新闻标题数据进行分类，并对给定文本进行预测。

二、 实验目的

掌握 Web 数据的获取、本地数据(txt、csv 格式)的读写，应用文本分类方法，对爬取到的新闻数据进行分类。

三、 实验内容

1. Web 数据爬取

从中国新闻网爬取对应分类的数据后，发现数据量太小。为增加数据集规模，在腾讯新闻（https://news.qq.com/）、新浪新闻（https://www.sina.com.cn/)DENG 等新闻网站 继续爬取数据。 并且，增加了清华大学中文文本分类数据集 THUCNews（http://thuctc.thunlp.org/）中相应标签下的数据。最终得到数据集规模如下：

- 训练集：17.5 万+
- 验证集：3.5 万+
- 测试集：3.5 万+

数据集中，各分类对应条目数量较为均衡。

```
url="http://www.chinanews.com.cn/cul/"
html=requests.get(url)
soup=BeautifulSoup(html.content,"html.parser")

news_1=[]
news=soup.find_all("li")+soup.find_all("em")+soup.find_all("h1")+soup.find_all("a")

for n in news:
    if n.a and n.a.string and len(n.a.string)>7:
        news_1.append(n.a.string)

print(len(news_1))

f=open("Data/2.txt","w")

for word in news_1:
    f.write(word+"\n")
```

2. 数据集使用

将数据集分别随机抽取 2%,, 5%, 10%, 20%, 50%, 100%, 使用不同规模数据集进行模型的训练和测试, 观察和对 比不同数据集规模下的模型性能。

3. 新闻标题分类

由于新闻标题数据没有特征, 也无法为数据集手动标注特征信息, 所以采用了不需要特征信息的**朴素贝叶斯**和能自动抽取特征信息的 Bert **神经网络**两种方法进行新闻标题分类。

3.1 朴素贝叶斯

朴素贝叶斯使用生成方法, 直接找出特征输出 Y 和特征 X 的联合概率分布 P(X,Y), 然后利用条件概率 P(Y|X) = P(X,Y)/P(X)计算

3.1.1 步骤

■ **导入数据**

将存储在 csv 中的新闻标题数据集导入为 pandas.Series 格式

```
data = pd.read_csv('Data/data2/3.csv', error_bad_lines=False)
data.head()
```

■ **划分训练集和测试集**

使用 sklearn 中的 model_selection 模块, 实现随机划分训练集和测试集

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(data['title'], data['tag'], random_state=1)
```

■ **分词**

使用 jieba 分词对新闻标题文本进行分词

```
def fenci(train_data):
    words_df = train_data.apply(lambda x:' '.join(jieba.cut(x)))
    return words_df
x_train_fenci = fenci(x_train)
x_train_fenci[:5]
```

■ **停词**

去除文本中没有实际意义的词语, 这里使用**哈工大中文停词表** (https://github.com/goto456/stopwords)

```
infile = open("hit_stopwords.txt",encoding='utf-8')
stopwords_lst = infile.readlines()
stopwords = [x.strip() for x in stopwords_lst]
```

■ **文本特征提取**

这里采用**词库表示法**, 通过 sklearn 中的 CountVectorizer 模块, 通过计数来将一个文档转换为向量。当不存在先验 字典时, Countvectorizer 作为 Estimator 提取词汇进行训练, 并生成一 CountVectorizerModel 用于存储相应的 词汇向量空间。该模型产生文档关于词语的稀疏表示。

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(stop_words=stopwords, max_features=5000)
```

```
vectorizer.fit(x_train_fenci)
```

■ **模型构建与评价**

使用 sklearn 中的 navie_bayes 模块，建立朴素贝叶斯分类器，并返回分类器在测试集上各个分类的准确率、召回率矩阵

```
from sklearn.naive_bayes import MultinomialNB
classifier = MultinomialNB()
#模型训练
classifier.fit(vectorizer.transform(x_train_fenci), y_train)
#使用训练好的模型进行预测
classifier.score(vectorizer.transform(fenci(x_test)), y_test)
```

■ **预测**

从测试文件中导入数据，使用先前训练好的朴素贝叶斯分类器进行预测。

```
f=open('test_7.txt','r',encoding='utf-8')

L=[]
m=f.readlines()
for i in m:
n=i.strip('\n')
x=n.split('\t')
L.append(x[1])
data_to_predict=pd.Series(L)
# data_to_predict=np.matrix(data_to_predict)
```

3.1.2 优化

1. TF-IDF 文本特征提取

词库表示法存在如下缺点：一些普遍出现的词，词频较高，看起来似乎是更重要的特征，但因为这个词普遍出现，这个词可能不是非常的重要。如果向量化特征仅仅用词频表示就无法反应这一点。
使用 TF-IDF（term frequency – inverse document frequency）进行文本特征提取，解决以上问题。
一个词语在一篇文章中出现次数越多，同时在所有文档中出现次数越少，越能够代表该文章.
下面直接给出一个词 x 的 TF-IDF 的基本公式如下：
TF(x) = (词 x 在文本中出现的次数) / (文本中的总词数)
IDF(x) = log((文本集合中的文本总数) / (包含词 x 的文本数))
TF-IDF(x) = TF(x) * IDF(x)

```
from sklearn.feature_extraction.text import TfidfVectorizer
#使用 tf-idf 把文本转为向量
tv = TfidfVectorizer(stop_words=stopwords, max_features=5000, lowercase = False)
tv.fit(x_train_fenci)
```

```
#模型训练
classifier.fit(tv.transform(fenci(x_train)), y_train)
#利用训练好的模型测试
# predict=classifier.predict(tv.transform(fenci(x_test)))
classifier.score(tv.transform(fenci(x_test)), y_test)
# print("每个类别的精确率与召回率",classification_report(predict, y_test))
```

2. N-gram 模型

在 N-gram 模型中，假设当前词的出现只与前面 N-1 个词有关，即将文本中的词或字符序列划分为连续的 N-1 个词或字符作为上下文，然后预测下一个词或字符。这个上下文可以用来计算条件概率，从而进行预测。

N-gram 模型的计算公式如下： P(w_n|w_1, w_2, ..., w_{n-1}) = Count(w_1, w_2, ..., w_n) / Count(w_1, w_2, ..., w_{n-1})其中，P(w_n|w_1, w_2, ..., w_{n-1}) 表示在给定上下文 w_1, w_2, ..., w_{n-1} 的条件下，预测下一个词 w_n 的概率。Count(w_1, w_2, ..., w_n) 表示在语料库中统计到的词序列 w_1, w_2, ..., w_n 的出现频率，Count(w_1, w_2, ..., w_{n-1}) 表示上下文 w_1, w_2, ..., w_{n-1} 出现的频率。

```
tv_2gram = TfidfVectorizer(stop_words=stopwords, max_features=5000, ngram_range=(1,2),lowercase = False)
tv_2gram.fit(x_train_fenci)
#训练模型
clf_2gram = MultinomialNB()
clf_2gram.fit(tv_2gram.transform(fenci(x_train)), y_train)
#预测
clf_2gram.score(tv_2gram.transform(fenci(x_test)), y_test)
```

2.2 Bert

使用朴素贝叶斯进行文本分类后，发现模型准确率始终在 80%左右，上网查阅资料后，了解到朴素贝叶斯网络计算量小，对于小规模数据集准确性高，但准确率上限确实没有深度学习模型高。于是想尝试使用深度学习模型，进一步提高本次实验的模型性能。这里选用了 Bert。

Bert（Bidirectional Encoder Representations from Transformers），是一个双向编码模型，采用了 Transformer Encoder block 进行连接， 在 11 种不同 NLP 测试中创出 SOTA 表现。

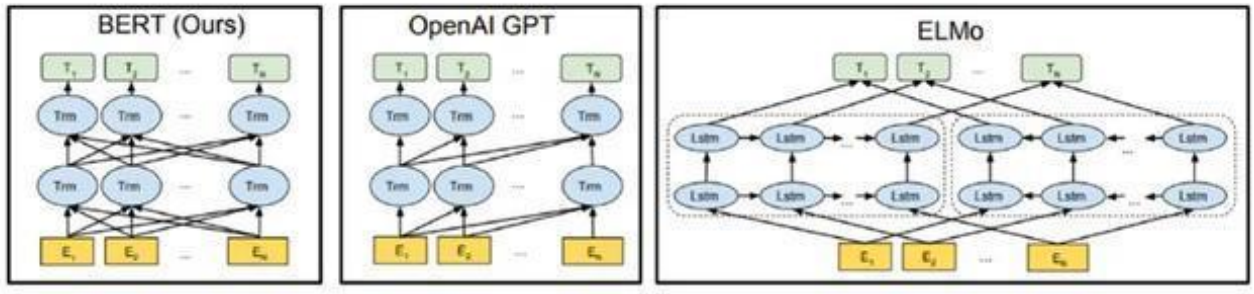


Figure 3: Differences in pre-training model architectures. BERT uses a bidirectional Transformer. OpenAI GPT uses a left-to-right Transformer. ELMo uses the concatenation of independently trained left-to-right and right-to-left LSTMs to generate features for downstream tasks. Among the three, only BERT representations are jointly conditioned on both left and right context in all layers. In addition to the architecture differences, BERT and OpenAI GPT are fine-tuning approaches, while ELMo is a feature-based approach.

四、 实验结果及分析

1. 朴素贝叶斯

1.1 不同数据集规模

对于总数据集中随机抽取的 2%,, 5%, 10%, 20%, 50%, 100%不同规模，使用朴素贝叶斯进行测试，观察和对比不同数据集规模下的模型性能。

1.1.1 2%规模

	precision	recall	f1-score	support
0	0.33	0.40	0.36	5
1	0.55	0.75	0.63	8
2	0.40	1.00	0.57	2
3	0.14	1.00	0.25	1
4	0.71	0.62	0.67	8
5	1.00	0.09	0.16	23
6	0.12	0.50	0.20	2
7	0.50	0.33	0.40	9

F1-score——0.39344262295081966

1.1.2 5%规模

	precision	recall	f1-score	support
0	0.43	0.80	0.56	15
1	0.70	0.59	0.64	27
2	0.86	0.33	0.48	36
3	0.50	0.37	0.42	19
4	0.72	0.54	0.62	24
5	0.57	0.92	0.71	13
6	0.45	0.82	0.58	11
7	0.67	0.64	0.65	22

F1-score——0.583563

1.1.3 10%规模

F1-score—0.7038269550748752

1.1.4 20%规模

F1-score——0.80228963938179

1.1.5 50%规模

F1-score——0.8152245322

1.1.6 100%规模

每个类别的精确率与召回率

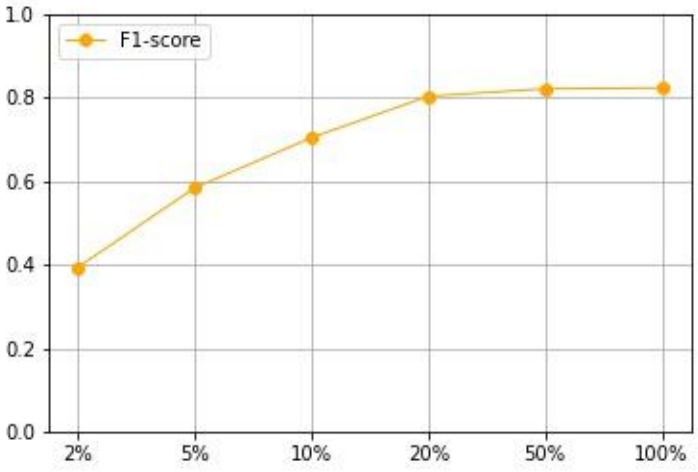
	precision	recall	f1-score	support
0	0.67	0.74	0.70	2651
1	0.71	0.77	0.74	2704
2	0.83	0.75	0.79	3493
3	0.89	0.83	0.85	9875
4	0.90	0.89	0.89	5623
5	0.84	0.85	0.84	8603
6	0.86	0.92	0.89	4947
7	0.29	0.41	0.34	218
accuracy			0.83	38114
macro avg	0.75	0.77	0.76	38114
weighted avg	0.84	0.83	0.83	38114

F1-score——0.8333420790260797

1.1.7 总结

总体来说，**时政**（0）、**国际**（1）、**生活**（7）类型的新闻标题，准确率和召回率较低。
分析：
可能因为时政新闻和国际新闻类型大致相同，只是一个在国内，一个在国外。
浏览在测试集上的分类结果，返现对于国际（1）类型的标注，很多情况下，在新闻标题中出现了外国国家名或者地名，就会把该条新闻划分到国际新闻种类下，但是有些新闻是关于财经、汽车或者体育的。
生活（7）类型的新闻标注准确率低，原因可能是生活新闻范围太大，包含内容过多。

以不同规模数据集进行模型训练得到的 F1-score 如图所示：



训练集规模小于 36000 条（总数据集的 20%），随着训练集的增大，分类模型准确率有较为显著的提升。大于 36000 条后，随数据集规模增大，模型准确率提升较小。
因此得出结论，在此模型中，训练数据集条目数在 30000~40000 之间，能较好的兼顾准确率与运行速度。

1. 2 优化

优化部分，对于各个规模的数据集（2%，5%，10%，20%，50%，100%）均做了测试，篇幅限制，这里只展示在 100%规模的数据集上的优化结果

1.2.1 TF-IDF 优化

每个类别的精确率与召回率

	precision	recall	f1-score	support
0	0.62	0.78	0.69	2322
1	0.61	0.84	0.71	2106
2	0.80	0.77	0.79	3256
3	0.91	0.79	0.85	10586
4	0.90	0.85	0.87	5816
5	0.85	0.82	0.84	9142
6	0.85	0.93	0.89	4876
7	0.02	0.50	0.03	10
accuracy			0.83	38114
macro avg	0.69	0.79	0.71	38114
weighted avg	0.84	0.83	0.83	38114

F1-score——0.8254709555543894

1.2.2 N-gram 模型优化

2-gram

每个类别的精确率与召回率

	precision	recall	f1-score	support
0	0.60	0.79	0.68	2224
1	0.61	0.84	0.70	2111
2	0.81	0.77	0.79	3297
3	0.91	0.78	0.84	10680
4	0.89	0.85	0.87	5800
5	0.85	0.82	0.84	9104
6	0.85	0.92	0.89	4889
7	0.02	0.56	0.03	9
accuracy			0.82	38114
macro avg	0.69	0.79	0.71	38114
weighted avg	0.84	0.82	0.83	38114

F1-score——0.8230046701999265

3-gram

每个类别的精确率与召回率

	precision	recall	f1-score	support
0	0.29	0.65	0.40	2543
1	0.33	0.72	0.45	1985
2	0.89	0.33	0.48	17272
3	0.33	0.60	0.43	2340
4	0.20	0.85	0.32	307
5	0.06	0.66	0.11	125
6	0.11	0.88	0.19	112
7	0.00	0.00	0.00	0
accuracy			0.43	24684
macro avg	0.28	0.59	0.30	24684
weighted avg	0.71	0.43	0.46	24684

0.44167497507477566

1.2.3 总结

使用 TF-IDF 方法优化后，F1-score 略微下降

使用 2-gram 方法优化后，F1-score 略微下降
使用 3-gram 方法优化后，F1-score 下降明显

五、 问题讨论

1. 关于朴素贝叶斯优化

分别使用 TF-IDF 和 2-gram 模型对朴素贝叶斯进行优化，但是实验结果发现，在所有的数据集规模下，优化后的模型准确率反而略微有所下降。3-gram 模型优化后，准确率显著下降。
使用 3-gram 模型后，准确率显著下降。这一点可以理解，引入过多上下文信息后，会造成准确率下降。
TF-IDF 和 2-gram 优化后，在各个数据集规模的训练中，准确率都下降。这一点着实不太理解。

2. 关于 Bert

最初使用 Bert 进行分类，结果很差，准确率只有 12%左右。一开始以为是数据集划分的原因，尝试了多种训练集、验证集、测试集划分比例，以及不同规模的训练数据后，准确率最高始终不超过 14%。
后来找到原因，实验中使用的 Bert 网络，调用了开源作者提供的预训练模型（https://github.com/649453932/Bert-Chinese-Text-Classification-Pytorch）。
发现问题后，尝试手动调整了参数和权重。但是，由于预训练模型中，文本的分类与此次实验，在类别数目和分类名称上有较大区别。在能看懂的范围内，无论怎样调整参数和权重，模型准确率始终处于较低水（16%~18%）。
之后开始尝试自己从头训练 Bert 网络，但是发现由于机器性能受限，训练进度非常非常非常慢，遂放弃 Bert 网络的方法。

六、 结论

1. 本实验中，朴素贝叶斯方法准确率（F1-score）最高为 0.83，Bert 方法准确率最高为 0.16。
2. **时政**（0）、**国际**（1）、**生活**（7）类型的新闻标题，准确率较低。
3. 本次实验的朴素贝叶斯模型中，训练数据集条目数在 30000~40000 之间，能较好的兼顾准确率与运行速度。